

Figure 1

The diagram illustrates the correlation between End-to-End Response Time and Quality of Experience. It shows a network architecture and a timeline of delays.

Network Architecture:

- Customer WAN (260):** Represented by a cloud icon.
- Customer Data Center LAN (242):** Represented by a cloud icon with server icons.
- Application Demarc (201):** A dashed line separating the customer network from the provider network.
- Network device (200):** A server icon.

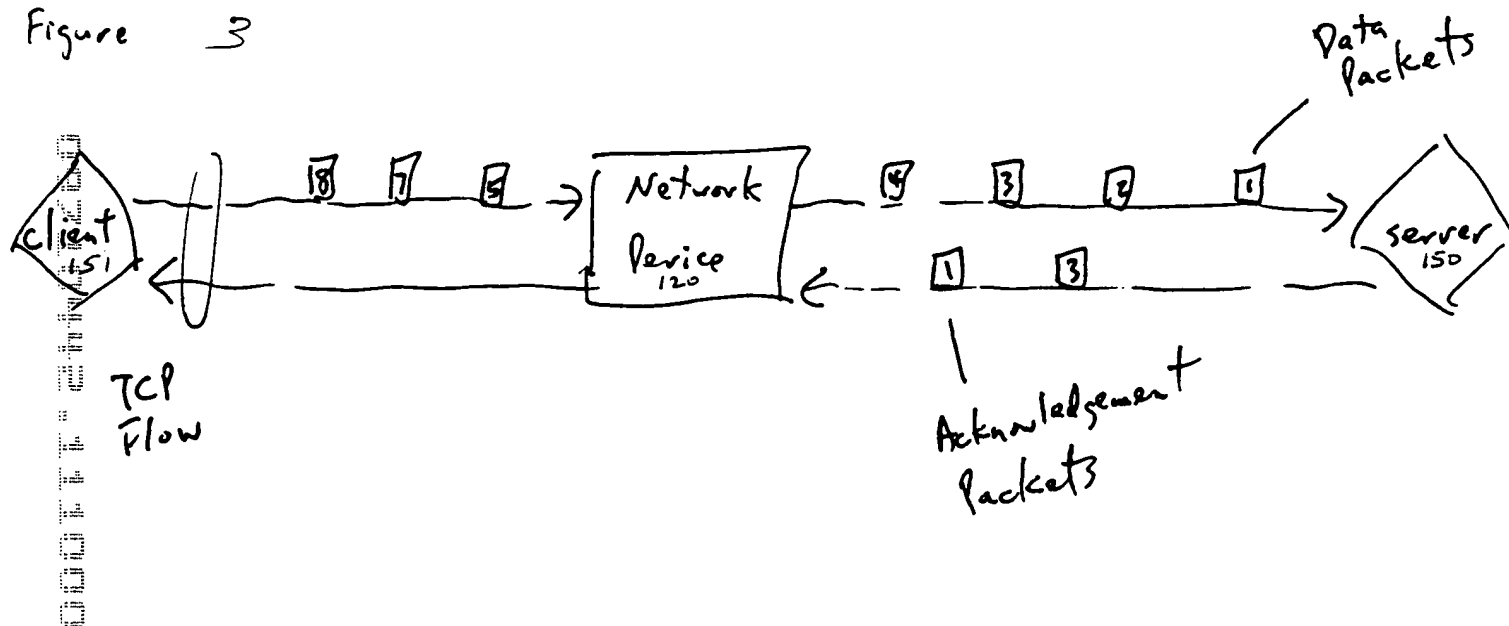
Timeline of Delays:

- Inbound Customer Network Delay:** The time taken for data to travel from the customer network to the provider network.
- Inbound Provider Network Delay:** The time taken for data to travel from the provider network to the host. A handwritten note "Host Latency" is next to this delay.
- Outbound Customer Network Delay:** The time taken for data to travel from the provider network back to the customer network.
- Outbound Provider Network Delay:** The time taken for data to travel from the provider network back to the host.

End-to-End Response Time - "Quality of Experience" Correlation: A horizontal arrow spanning the entire duration of the delays, indicating the total time experienced by the user.

Figure 2

Figure 3



SCANNED, #
drixoral.c

Figure 4

```

1 /.....
2 *
3 * Module Name: drixoral.c
4 *
5 * Abstract: Congestion Index Measurement
6 * Author: Guy Biddle
7 * Created: 991102
8 *
9 * $Revision: 1.1.4.6 $ $Name: $
10 * $Date: 2000/03/22 22:20:24 $ Copyright Packeteer, Inc. 1999
11 .....
12 */
13
14 #include <artist.h>
15 #include <mecom.h>
16 #include <kmalloc.h>
17 #include "../tcp.h"
18
19 #if _PSAPPV
20
21 #define MY_DEBUG_LEVEL gtcprtmdebuglevel
22
23 extern int gtcprtmdebuglevel;
24
25 #define metClassCongestionAccum(tc, msec) \
26 metClassPktExchMsecsAccum(tc, msec)
27
28 #define DRIX_SEQ_SPOTS 7 /* chosen to avoid kmalloc roundup wastage */
29
30
31 typedef struct {
32     SEQ lastseq;
33     SEQ lastack;
34     SEQ seq[DRIX_SEQ_SPOTS];
35     TICK time[DRIX_SEQ_SPOTS];
36 } CongestionMeasurement, *CongestionMeasurementPtr;
37
38 void
39 drixoralMeasureCongestion(
40     TCB_INFO_PTR info
41 ) {
42     TCB_PTR tcb;
43     ArtMeasurementPtr am;
44     CongestionMeasurementPtr drix;
45     TCLASS_PTR tc;
46     DIRECTION dir;
47     SEQ seq;
48     UINT12 msec;
49     int i, leave;
50
51     whereStr("xooMeasureCongestion");
52
53     tcb = info->tcb;
54     am = tcb->artData;
55     dir = info->dir;
56
57     blurt3("seq %d ack %d flags %X", info->header.seq, info->header.ack, info->header.flags);
58     blurt4("dir %d len %d end %d tcb %X", dir, info->header.dataLen, info->header.seq + info
59     ->header.dataLen, tcb);
60
61     if (info->header.dataLen > 0) {
62         if (!am->drixoral(dir))
63             am->drixoral(dir) = kcalloc(sizeof(CongestionMeasurement), M_DRIXORAL);
64
65         if ((dirx = am->drixoral(dir))) {
66             seq = info->header.seq + info->header.dataLen;
67             if (info->header.flags & (TCP_F_SYN|TCP_F_FIN))
68                 seq++;
69
70             if (SEQ_GT(seq, drix->lastseq) || !drix->lastseq) {
71                 .or(i = 0; i < DRIX_SEQ_SPOTS; i++)
72                     if (!drix->seq[i])
73                         break;
74

```

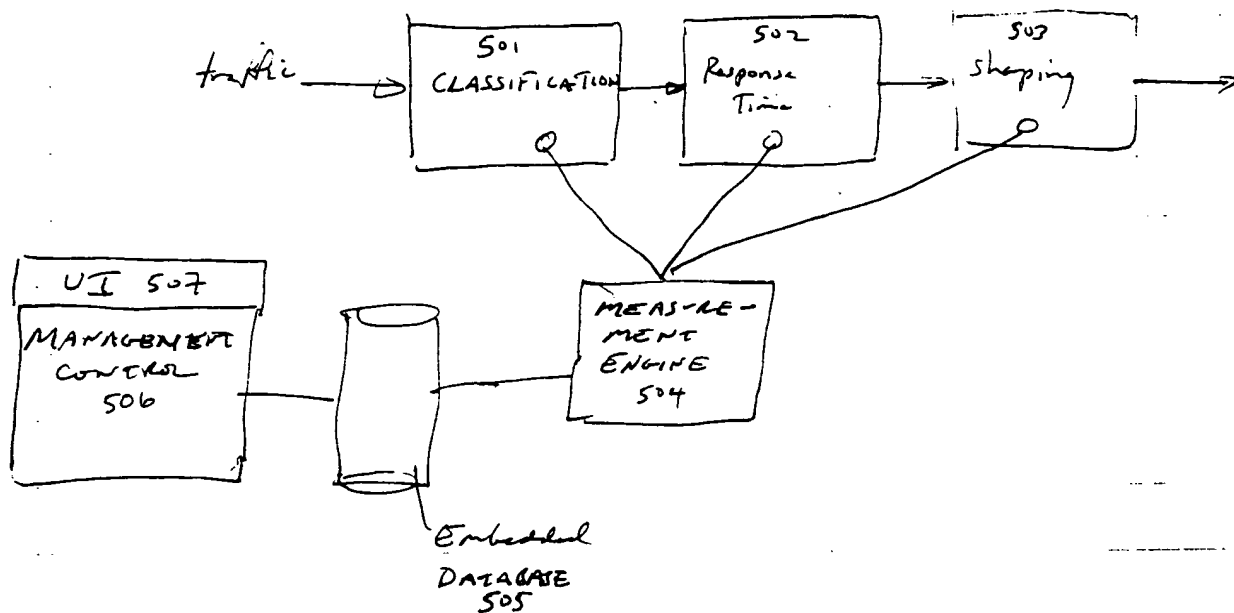
```

73 if(i < DRIX_SEQ_SPOTS){
74     drix->seq[i] = seq;
75     drix->time[i] = info->bcb->tick.ticks;
76     }else
77     {
78         info2("req td spot overflow tcb %X", seq, tcb);
79     }
80     drix->lastseq = seq;
81     }
82     }
83     }
84     }
85     dlr = OTHER_DIRECTION(info->dir);
86     if(IS_FLAG_SET(info->header.flags, TCP_F_ACK) && (drix = am->drixoral(dlr))
87     && (SEQ_GT(info->header.ack, drix->lastack) || !drix->lastack)){
88         leave = 0;
89     }
90     for(i = 0; i < DRIX_SEQ_SPOTS; i++){
91         if(!drix->seq[i])
92             continue;
93         else if(!SEQ_EQ(info->header.ack, drix->seq[i])){
94             if(info->header.dataLen == 0){
95                 msec = TICKS_TO_MSECS_ROUNDED(info->bcb->tick.ticks - drix->time[i]);
96                 if((tc = tcClassIdToTClassFast(tcb->halfConn[dlr].gear.classId)){
97                     metClassCongestionAccum(tc, msec);
98                     blurt4("sample td msec td class %a dir %d", i, msec, tc->name, dlr);
99                     }else
100                     attni("no class for TCB %X", tcb);
101                 }
102                 drix->seq[i] = 0;
103             }else if(SEQ_GT(info->header.ack, drix->seq[i])){
104                 drix->seq[i] = 0;
105                 leave++;
106             }
107             if(!leave)
108                 drix->lastack = info->header.ack;
109             else{
110                 kfree(drix);
111                 am->drixoral(dlr) = 0;
112             }
113             }
114             void
115             drixoralCleanup(
116                 TCB_PTR tcb
117             ){
118                 ArtMeasurementPtr am;
119                 CongestionMeasurementPtr drix;
120                 am = &tcb->artData;
121                 if((drix = am->drixoral(DIR_INBOUND))){
122                     kfree(drix);
123                     am->drixoral(DIR_INBOUND) = 0;
124                 }
125                 if((drix = am->drixoral(DIR_OUTBOUND))){
126                     kfree(drix);
127                     am->drixoral(DIR_OUTBOUND) = 0;
128                 }
129                 }
130                 }
131                 }
132                 }
133                 }
134                 }
135                 }
136                 }
137                 }
138                 }
139                 }
140                 }
141                 }

```

Send it to: "P&P" /

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99



✓
figure 5